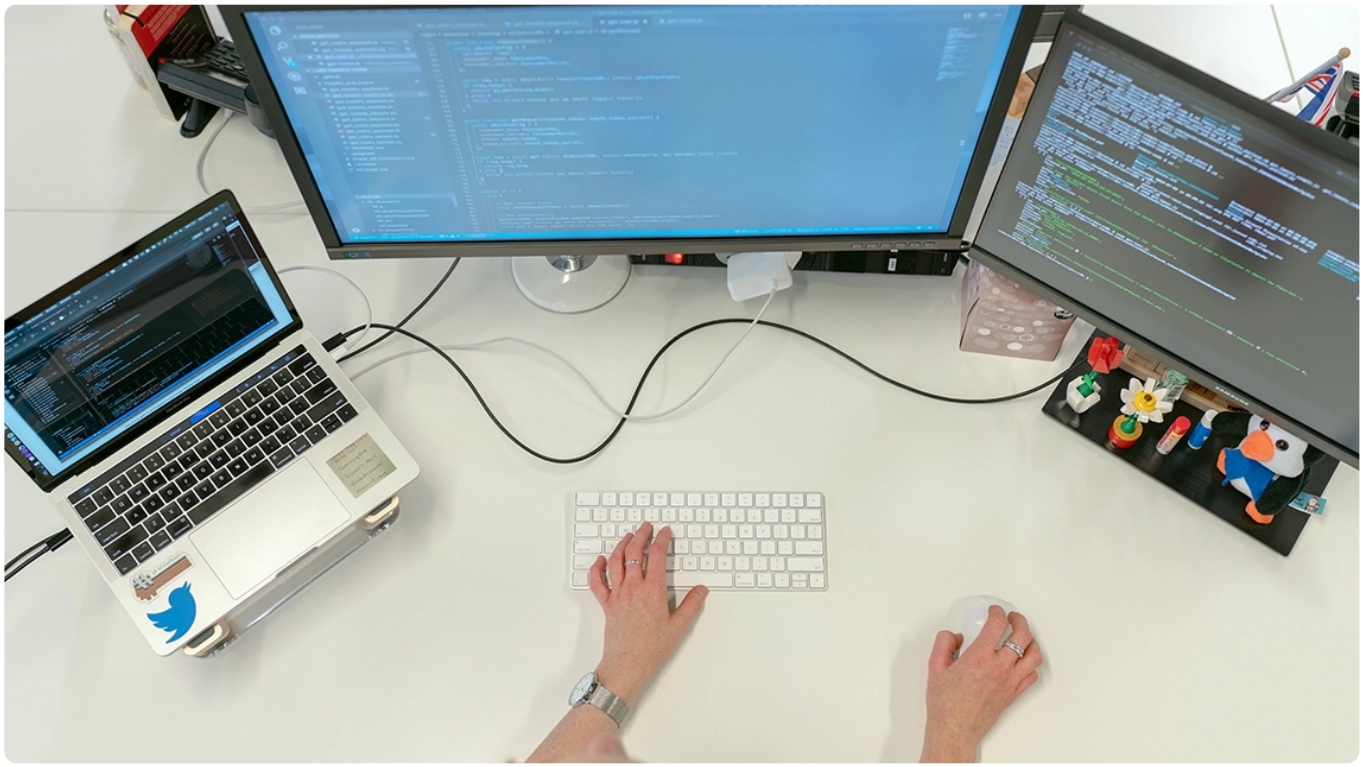
08｜化骨绵掌：降级、热点和容灾处理

2021-10-13 佘志东

《手把手带你搭建秒杀系统》 课程介绍 >



讲述：佘志东
时长 18:06 大小 16.58M

你好，我是志东，欢迎和我一起从零打造秒杀系统。

上节课我们介绍了秒杀的削峰，你在手写秒杀系统的时候，可以采用验证码 / 问答题、异步消息队列或者限流的方式进行削峰，以此平滑流量峰值，减轻单位时间分片内的系统压力。这节课我们将把重点放在其他高可用的方面——降级、热点数据和容灾，**持续打造秒杀系统的高可用**。

当秒杀活动开启，流量洪峰来临时，交易系统压力陡增，具体表现一般会包括 CPU 升高、IO 等待变长，请求响应时间 TP99 指标变差，整个系统变得越来越不稳定。为了力保核心交易流程，我们需要对非核心的一些服务进行降级，减轻系统负担，这种降级一般是有损的，属于“弃卒保帅”。

而秒杀的核心问题，是要解决单个商品的高并发读和高并发写的问题，这是典型的热点数据问题，我们需要有相应的机制，避免热点数据打垮系统。

机房容灾其实不仅仅是秒杀系统需要思考的，重要的软件系统，不管是互联网应用，还是传统应用，比如银行系统等，都需要考虑机房容灾的问题。不同的场景，容灾的设计也不尽相同，这节课我们将从常见的互联网公司的角度，看看他们一般会怎么搭建交易系统的容灾。

降级

我们先说说“降级”，其实和削峰一样，降级解决的也是有限的机器资源和超大的流量需求之间的矛盾。如果你的资源够多，或者你的流量不够大，就不需要对系统进行降级了；只有当资源和流量的矛盾突出时，我们才需要考虑系统的降级。

前面已经介绍了，降级一般是有损的，那么必然要有所牺牲，下面介绍几种常见的降级：

写服务降级，牺牲数据一致性获取更高的性能；

读服务降级，故障场景下紧急降级快速止损；

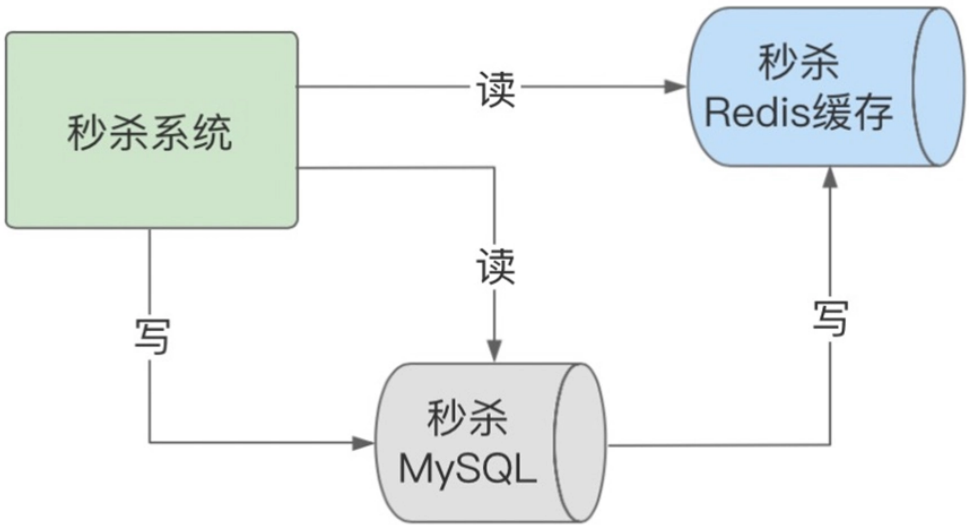
简化系统功能，干掉一些不必要的流程，舍弃非核心功能。

下面我们逐一分析下。

1. 写服务降级，牺牲数据一致性获取更高的性能

我们知道，在多数据源（MySQL 和 Redis）的场景下，数据一致性一般是很难保证的。除非你引入分布式事务，但分布式事务也会带来一些缺点，比如实现复杂、性能问题、可靠性问题等。因此一般在涉及金融资产类对一致性要求高的场景时，我们才会考虑分布式事务。

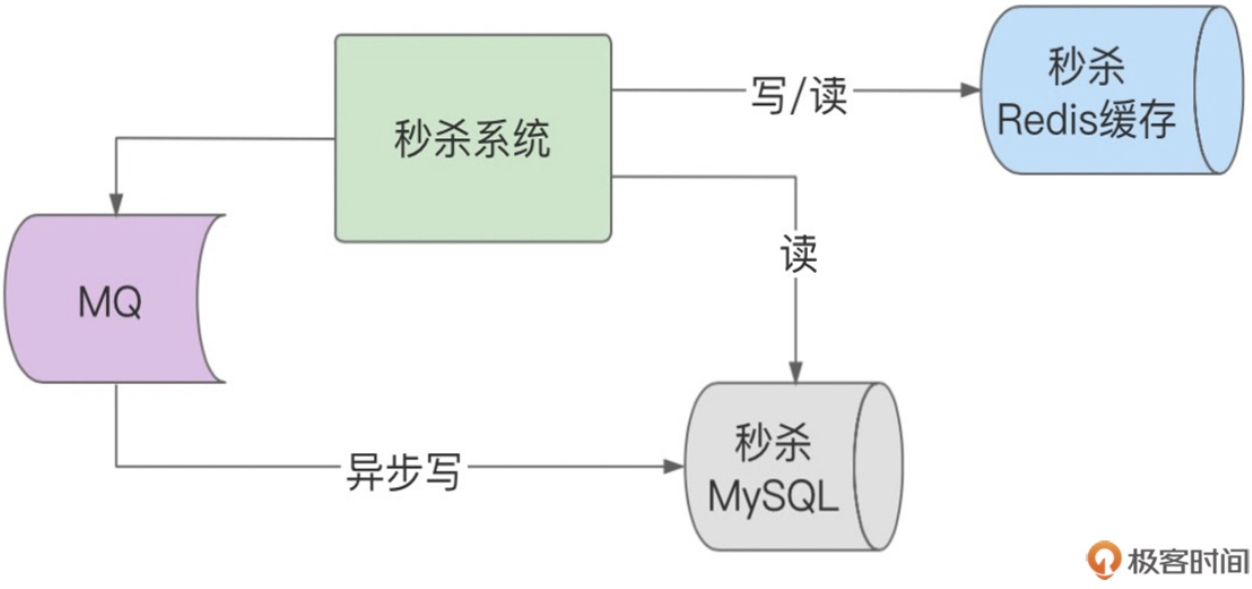
在流量不高的时候，我们的写请求可以直接先落入 MySQL 数据库，再通过监听数据库的 Binlog 变化，把数据更新进 Redis 缓存，如下图所示。



这种设计，缓存和数据库是最终一致的。通过缓存，我们可以扛更高流量的读操作，但是写操作仍然受制于数据库的磁盘 IOPS，一般考虑一个数据库也就能支持 3000～5000 TPS 的写操作。

当流量激增的时候，我们就需要对以上的写路径进行降级，由同步写数据库降级成同步写缓存、异步写数据库，利用 Redis 强大的 OPS 来扛流量，一般单个 Redis 分片可达 8～10 万的 OPS，Redis 集群的 OPS 就更高了。

如下图所示，写请求首先直接写入 Redis 缓存，写入成功之后，同时再启动一个线程，发出写操作 MQ，就可以返回客户端了。其他应用消费 MQ，通过 MQ 异步化写数据库。

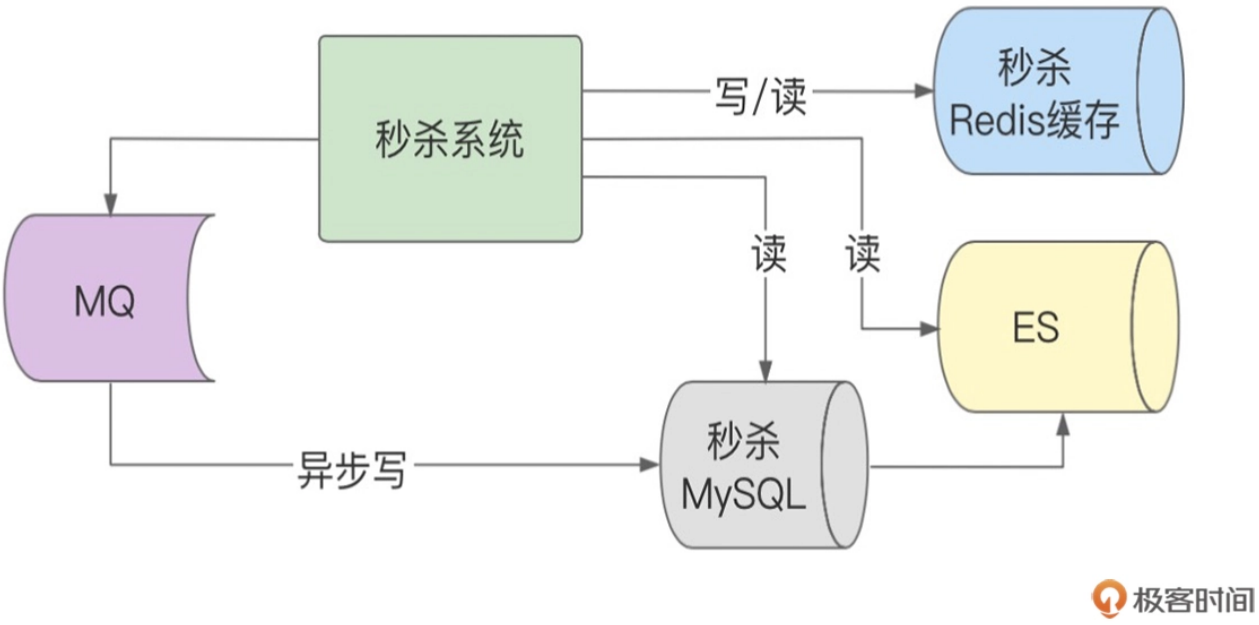


这里，我们通过 Redis 的高并发写能力，提升了系统性能，带来的牺牲就是缓存数据和数据库数据的一致性问题。为了追求高性能，牺牲一致性在大厂的设计中比较常见，对于异步造成的数据丢失等一致性问题，一般会有定时任务一直在比对，以便最快发现问题，进行修复。

2. 读服务降级，故障场景下紧急降级快速止损

在做高可用系统设计时，我们都会有个共识，就是微服务自身所依赖的外部中间件服务或者其他 RPC 服务，随时都可能发生故障，因此我们需要建设多级缓存，以便故障时能及时降级止损。

如下图所示，我们给 Redis 缓存之外，又增加了 ES 缓存。当然了，你可以建立多个缓存副本，比如主 Redis 缓存外，再建立副 Redis 缓存，或者再增加 ES 缓存，这些都可以的，不过相应会增加你的资源成本和代码编写的复杂度。



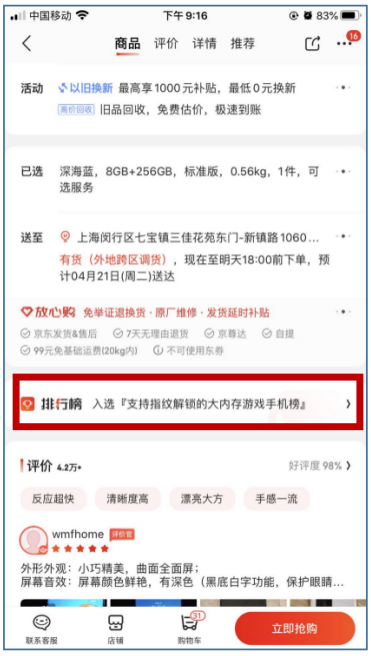
如上图，假设当秒杀的 Redis 缓存出现故障时，我们就可以通过降级开关，快速将读请求降级到 ES 上。或者当 Redis 和 ES 同时出现故障时（现实中很少出现同时故障的场景），我们还是可以通过降级开关将流量切换到数据库上，让数据库暂时承压来完成读请求服务。

由此可见，在做高可用系统设计时，降级路径是多么的重要，它会是你关键时候的保命开关，让你在突发故障时有路可退。

3. 简化系统功能，干掉一些不必要的流程，舍弃非核心功能

当你打开京东或淘宝的商品详情页时，你会发现，除了商品的基本信息外，还有很多附加的信息，比如你是否收藏过该商品、商品的收藏总数量、商品的排行榜、评价和推荐等楼层。同样，对于秒杀结算页，还会有礼品卡、优惠券等虚拟支付路径。

如果是普通商品，这些附加信息当然是越多越好，一方面体现了系统的完整性，另一方面也可以多渠道引流促进转化。但是在秒杀场景下，这些信息是否有必要就需要视情况而定了，秒杀系统要求尽量简单，交互越少，数据越小，链路越短，离用户越近，响应就越快，因此非核心的功能在秒杀场景下都是可以降级的，如下图红框所示。

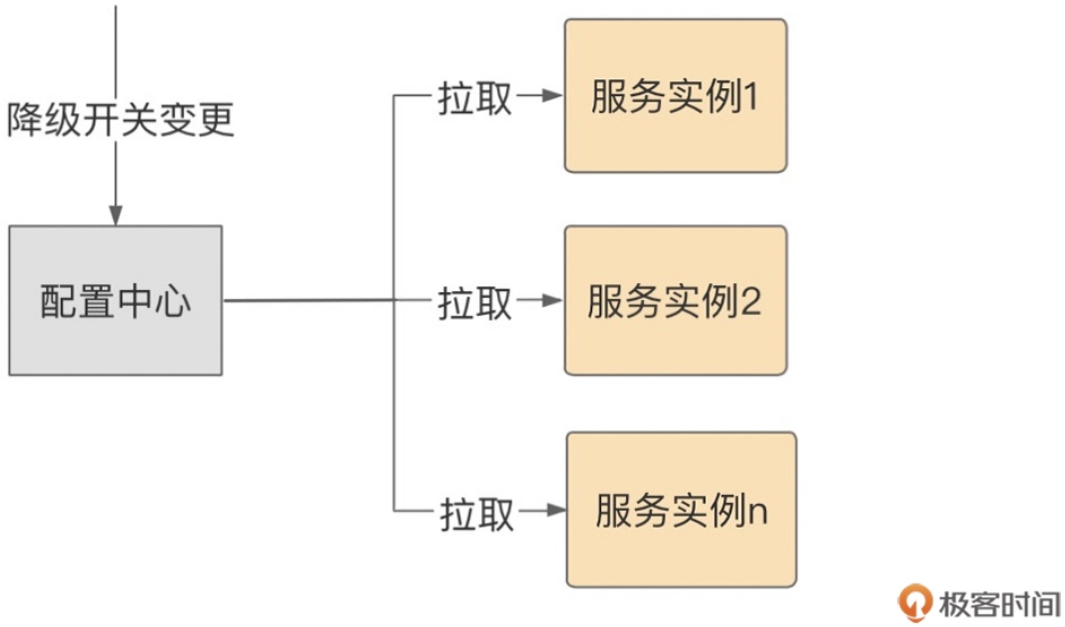


这种非核心功能的有损降级，要视具体的 SKU 而定，一般为了降低影响范围，我们只对流量非常高的 SKU 进行降级。比如，如果是手机秒杀，一般是不需要降级的，但是像茅台、口罩这样的爆品，就需要针对 SKU 维度进行非核心功能的降级了。

以上就是几个典型的降级场景了，简单总结一下。这 3 种降级场景在秒杀系统建设中都会用到。首先，非核心链路的降级在爆品的 SKU 上经常用到，你可以收藏一下京东的平价茅台商品，然后进入商详页看下是否有收藏功能，就会发现一直是处于降级的状态；非核心链路的降级在大促时也经常用，一般在快接近 0 点前的 5 分钟，全系统就会启动很多非核心功能的降级，以确保有限的机器资源用在更核心的场景。故障场景的读服务降级也是常用的高可用手段，通过一键降级开关我们可以灵活的在不同链路间进行切换，提供灵活的

服务能力，应对可能突发的故障。为了追求高性能，秒杀也会牺牲一致性对写服务进行降级。

这里我们顺便也看下**降级开关的设计**，比较简单，核心思路就是通过配置中心，对降级开关进行变更，然后推送到各个微服务实例上。



热点数据

讲完了降级，接下来我们来聊聊热点数据。进入正题前，我们先看看高并发的常规解决思路。

分布式系统设计，解决高并发问题，可能你很快会想到，如果是数据库，可以通过分库分表来应对，如果是 Redis，可以增加 Redis 集群的分片来解决，而应用层一般是无状态的设计。所以从数据库、Redis 缓存到应用服务，都是可以通过增加机器来水平扩展服务能力，解决高并发的问题。

然而，这样就能应对秒杀的挑战了吗？其实还不够，前面我有提到，秒杀的核心问题是要**解决单个商品的高并发读和高并发写问题，也就是要处理好热点数据问题。**

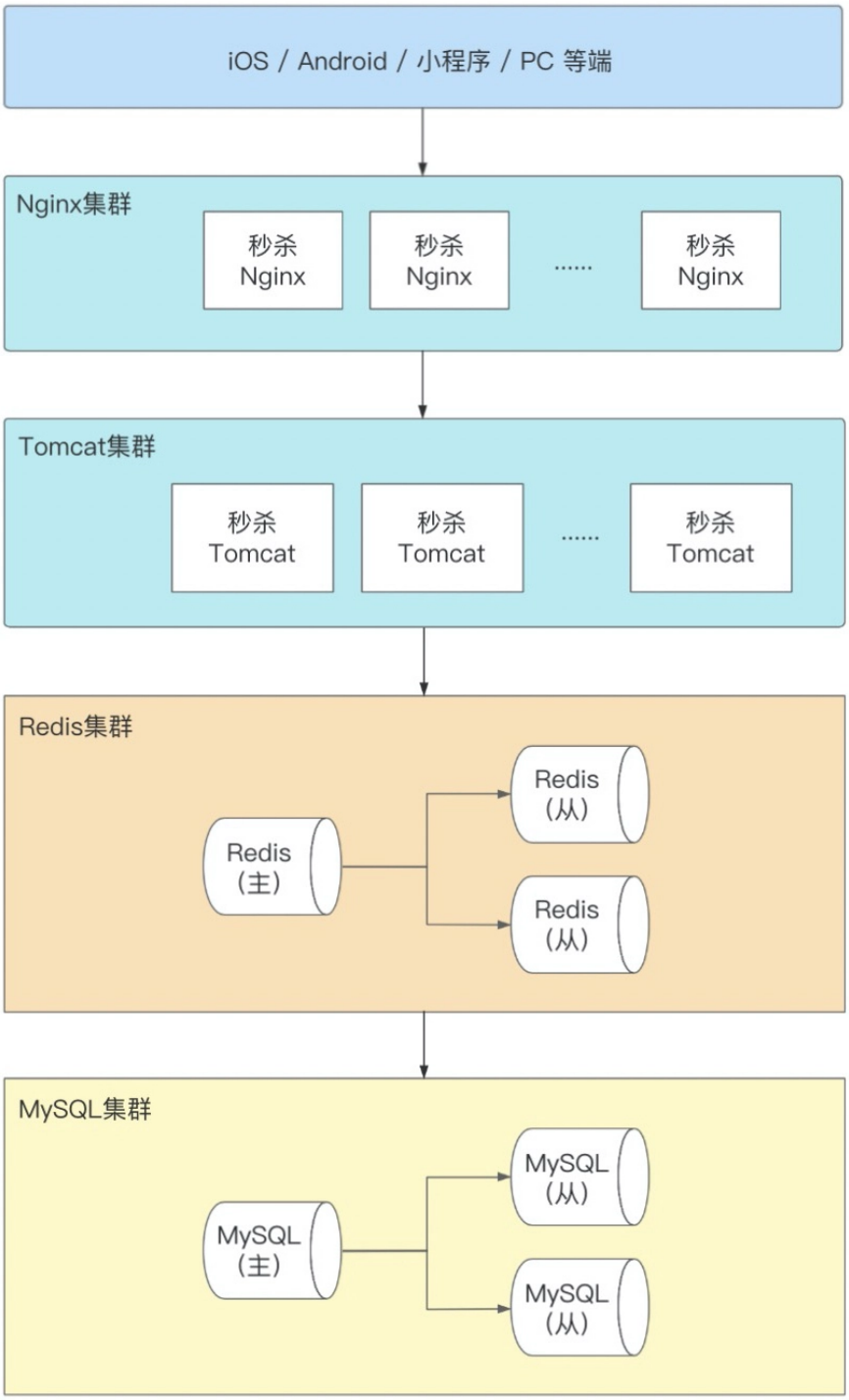
所谓热点数据，是从单个数据被访问的频次角度去看的。单位时间（1s）内，一个数据非常频繁的被访问，就可以称之为热点数据，反之可以归为一般数据或冷数据。那么单位时间内究竟多高的频次才能称为热点数据呢？实际上并没有一个明确的定义，可以根据你自己的系统吞吐能力而定。

平价茅台在进行秒杀时，只有这个 SKU 是热点，所以再怎么进行分库分表，或者增加 Redis 集群的分片数，茅台 SKU 落在的那个分片的能力实际并没有提升，总会触达上限，把 Redis 打挂，最后可能引发缓存击穿、系统雪崩。那我们应该怎么解决这个棘手的热点问题呢？别担心，难不倒我们，请跟我继续往下学习。

我们把这个问题分为两类：读热点问题和写热点问题。下面我们分别展开讨论。

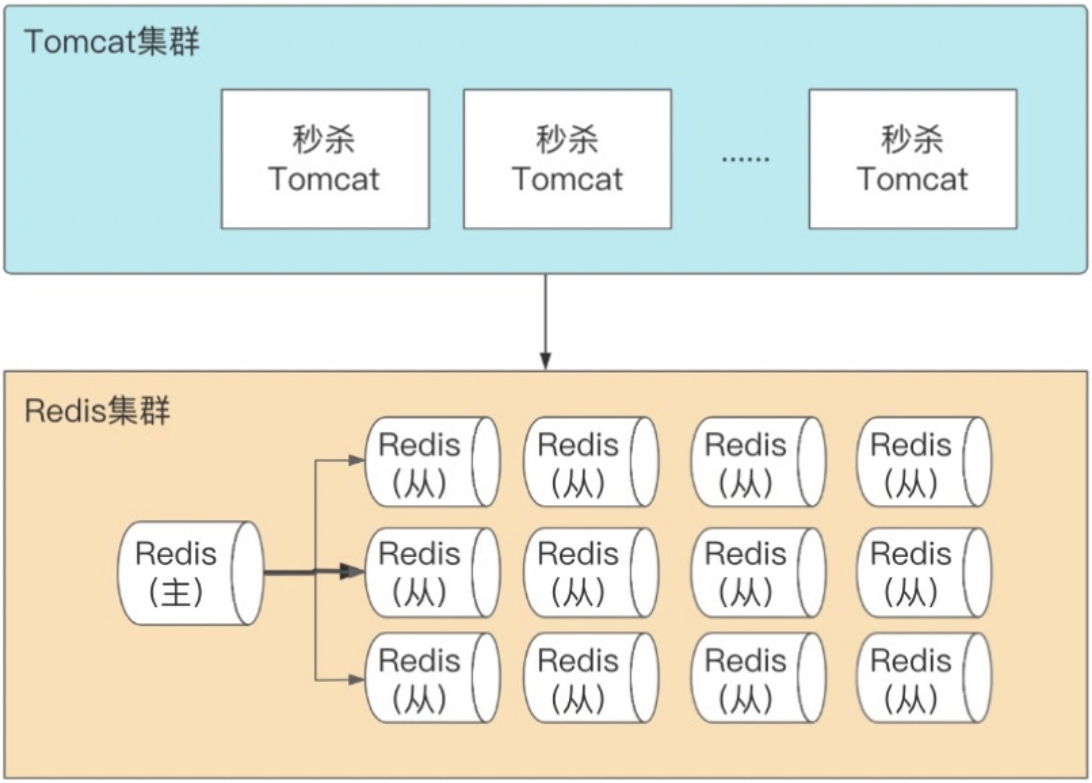
先看下**读热点**如何解决，我先抛出解决该问题的思路：

1. 增加热点数据的副本数；
2. 让热点数据离用户越近越好。



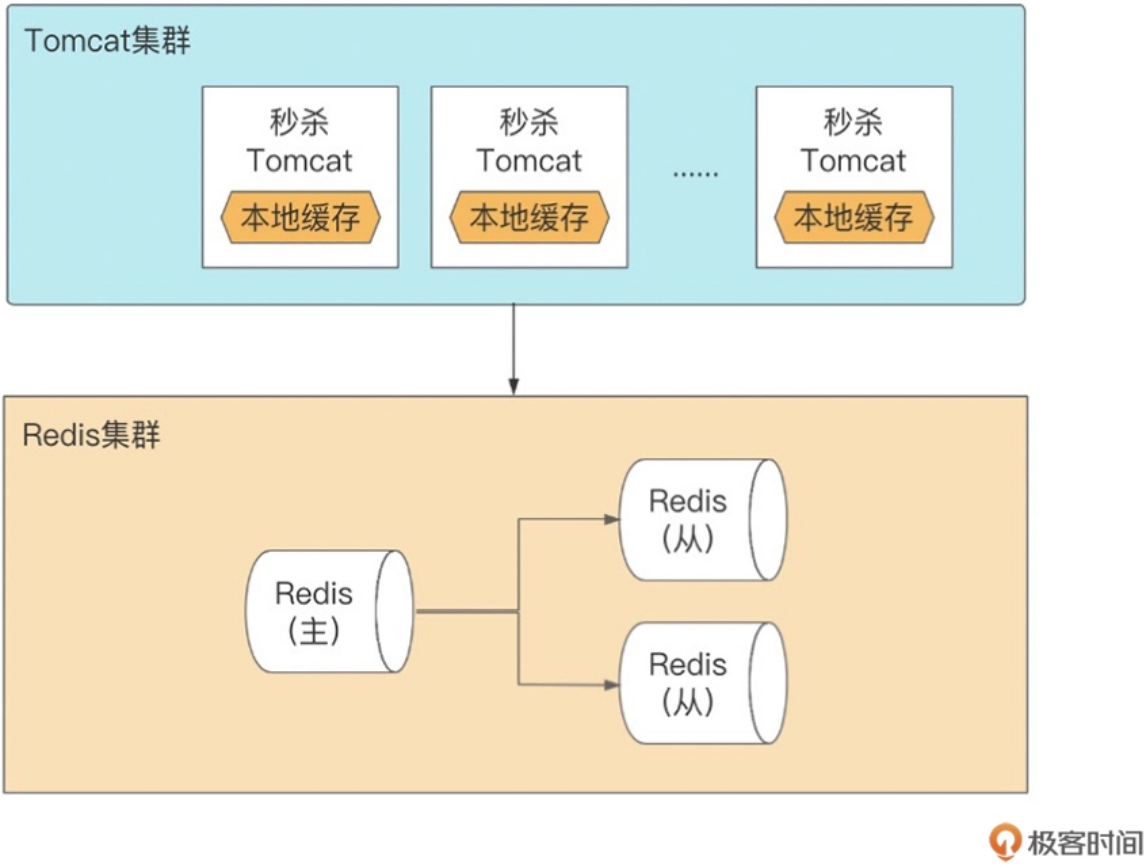
极客时间

以上是秒杀系统的部署结构图，参照解决思路，我们的第一个解决方案，就是**增加 Redis 从的副本数**，然后业务层（Tomcat 集群）轮询查询不同的副本，提高同一数据的 QPS。一般情况下，单个 Redis 从，可提供 8~10 万的查询，所以如果我们增加 12 个副本，就可以提供百万 QPS 的热点查询。



这个方法能解决热点问题，但成本比较高，如果你的集群分片数比较多，那分片数 * 副本数就是一笔不小的开销。

第二个解决方案，我们**把热点数据再上移**，在 Tomcat 集群做热点数据的本地缓存，也就是让业务层的每个实例里都有份数据副本，读请求数据的时候，无需去 Redis 获取，直接从本地缓存里取。这时候，数据的副本数和 Tomcat 实例一样多，另外请求链路减少了一层，而且也减少了对 Redis 单片 QPS 上限的依赖，具有更高的可靠性和更高的性能。



这种方式热点数据的副本数随实例的增加而增加，非常容易扩展，扛高流量。不过你要思考一个问题，本地缓存的数据延迟业务是否能够接受？

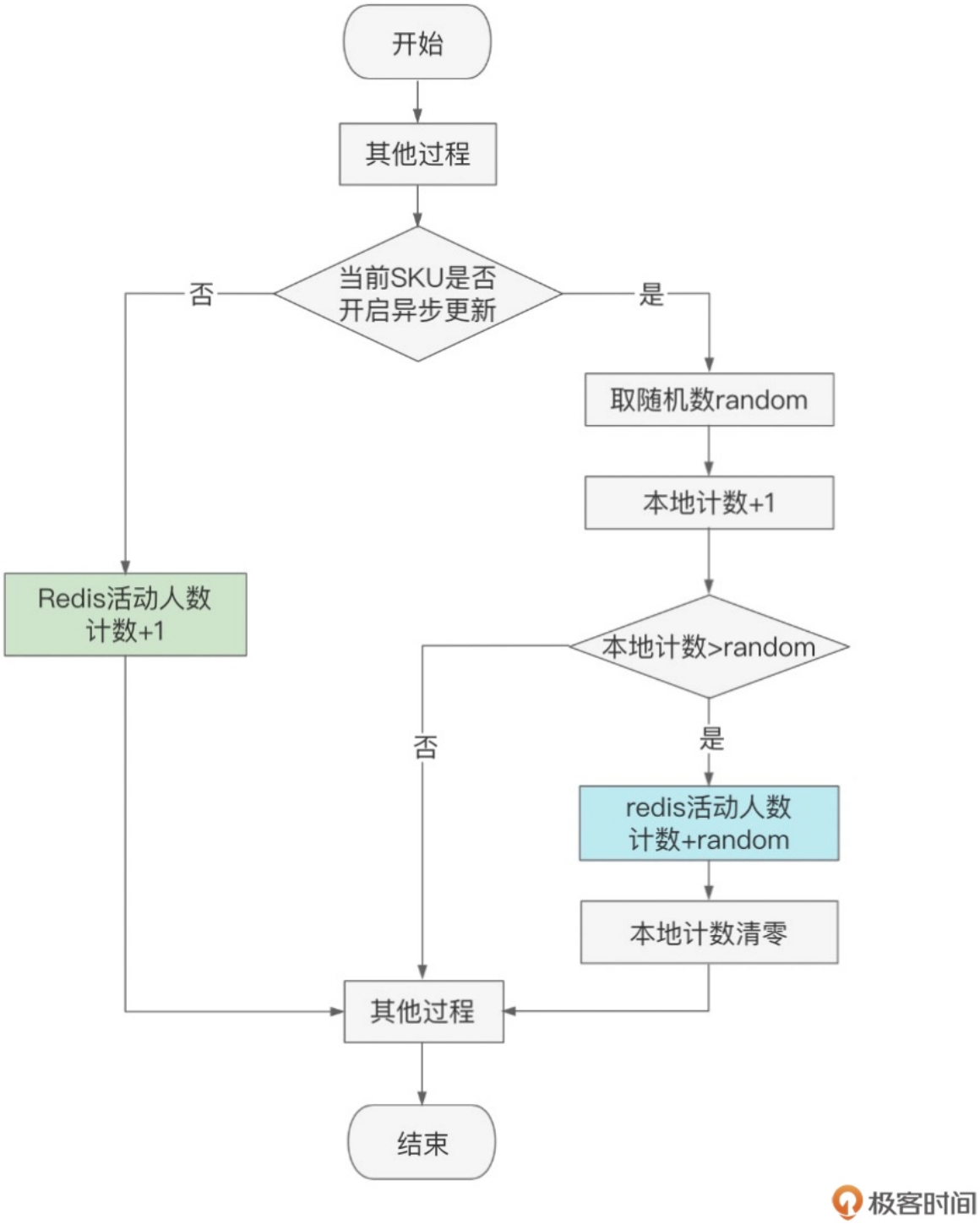
如果能接受，本地缓存的时候可以设置几分钟？如果对延迟要求比较高，可以设置 1s，这样对 Redis 而言，OPS 的压力直接降低到实例数 / 每秒，就不需要那么多副本了。

本地缓存的实现比较简单，可以用 HashMap、Ehcache，或者 Google 提供的 Guava 组件。

读热点还有一个比较简单粗暴的方法，那就是直接短路返回。这么说可能比较抽象，我举个例子，茅台秒杀的时候，这个 SKU 是不支持使用优惠券的，那么优惠券系统在处理的时候，可以根据配置中心的茅台 SKU 编码，直接返回空的券列表，这样基本上不怎么耗资源，效率非常高。当然了，这种方式和具体商品的活动方式有关，不具有通用性，但是在几百万的流量面前，简单有效。

介绍完读热点，接下来我们看**写热点**问题。我们先回忆一下，在第 6 讲流量管控里，我们介绍到用户点击“立即预约”的时候，会往“预约人数”这个 Redis key 上进行 ++ 操作，当几百万人同时预约的时候，这个 key 就是热点写操作了。

这个预约总人数有个特点，只是在前端给用户展示用，除此之外，没有其他用途，因此在高并发的场景下，这个人数可以不用那么及时和精确。知道了问题所在，解决方案就在眼前了，我们的思路就是先在 JVM 内存里 ++，延迟提交到 Redis，这样就可以把 Redis 的 OPS 降低几十倍。以下是示意图：



写热点还有一个场景就是库存的扣减，这里讲一下基本思路，**可以通过把一个热 key 拆解成多个 key 的方式，避免热点问题**。这种设计涉及到对库存进行再细分，以及子库存挪

动，非常复杂，而且边界问题比较多，容易出现少卖或者超卖问题，一般不推荐这种方法。

另一个思路就是对单 SKU 的库存直接在 Redis 单分片上进行扣减，实际上，库存系统在秒杀链路的末端，通过我们之前介绍的削峰和限流，真正到库存的流量是有限的，单片的 Redis OPS 能承受得了。然后，我们可以针对单 SKU 的库存扣减进行限流，保证库存单片 Redis 的压力。这样双管齐下，单 SKU 的库存 Redis 扣减压力就是可控的了。

容灾

最后我们一起看下容灾，容灾不仅仅是秒杀系统需要考虑的，但凡重要的系统，都要在方案设计时考虑容灾问题。容灾，一般是指搭建多套（两套或以上）相同的系统，当其中一个系统出现故障时，其他系统能快速进行接管，从而持续提供 7*24 不间断业务。

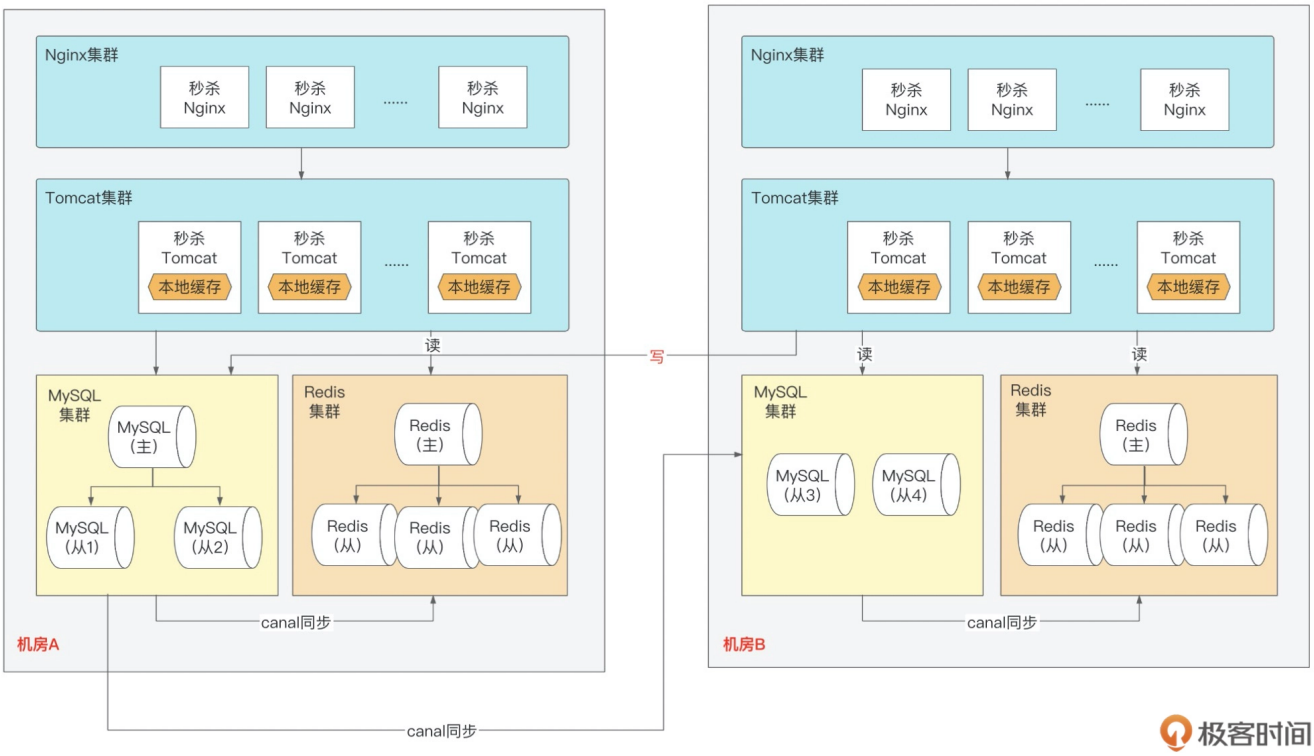
在讨论容灾的时候，你可能听说过“同城双活”“异地多活”等术语，它们都是不同的容灾方案，不同的方案，其技术要求、建设成本、运维成本都不一样。在多活架构下，对两套系统之间通信线路质量、时延要求很高，业内主流 IT 厂家比较认可的是单向时延 2ms 以内，超过这个时延，对“多活”的跨机房请求和数据同步的性能影响就会比较大。

因此，涉及跨城市的多活，当城市距离较大时，比如上海和北京，那么这种物理上的时延很难克服。为了保证数据库的一致性，就需要付出很高的时间成本，往返几个来回时延叠加，RT 就受不了了。所以，如果是异地多活的情况，一般是需要把数据划分成不同单元，让流量在单元内闭环。异地多活单元化的设计其实非常复杂，成本高昂，即便是大厂也不一定能搭建好异地多活。

因此，这节课我们的重点还是放在“同城双活”的设计上。

同城双活是在同城或相近区域内建立两个机房。同城双机房距离比较近，通信线路质量较好，比较容易实现数据的同步复制，保证高度的数据完整性和数据零丢失。

同城两个机房各承担一部分流量，一般入口流量完全随机，内部 RPC 调用尽量通过就近路由闭环在同机房，相当于两个机房镜像部署了两个独立集群，数据仍然是单点写到主机房数据库，然后实时同步到另外一个机房。



如上图所示，就是秒杀系统的“同城双活”方案。从 Nginx 层、Tomcat 层，到 Redis、MySQL 层，我们都做了双中心部署，不管哪一层出现故障，都可以灵活切换。

同城双活因为物理距离短，机房间的时延是有保证的，我们可以让写流量最后落库的时候都写到主机房，而读流量则完全可以做到机房内闭环。当然了，我们在做系统设计的时候，也是要尽量避免 C 端流量直接打到数据库，因此，这种跨机房的写流量都是比较可控的。

简单提示一下，双机房间的物理专线也必须是高可用的设计，至少需要两根以上进行互备，这样在专线故障时才有机会绕行避免不可用，这些在大厂里一般是运维团队在保障，业务团队了解实现原理就可以。

小结

这节课我们主要讨论了秒杀的降级策略，热点数据的处理方式以及“同城双活”的容灾方案。

降级是系统故障发生时你的逃生路径，你一定要有这个认知。系统故障不可避免，随时都可能发生，所以在做系统设计时一定要给自己预留逃生通道，不能在系统故障时让用户只能干等着故障恢复。

所以降级的设计非常重要，这一节课里，我们介绍了几种常见的降级场景和解决方法，有同步写库降级为异步写库，其实也可以反过来，从异步写库降级为同步写库，取决于你追求的是性能还是一致性；我们还介绍了通过搭建多级缓存，在一级缓存故障时就可以降级到二级缓存；最后我们还介绍了业务功能降级，舍弃非核心功能，力保主流程功能正常运转。

当我们有了降级手段后，日常就要经常演练了，避免线上真的发生故障时茫然失措。

接着我们还介绍了秒杀的热点数据处理，热点数据是秒杀系统的基本属性，必须面对。读热点问题的解决遵循朴素的思路，通过增加数据副本数来扛流量，同时尽量让数据靠近用户。这节课我们更多着墨在动态热点数据上，通过搭建 Redis 多从副本以及 JVM 本地缓存，能解决大部分的读热点问题；而对于静态数据的处理，可以通过 CDN 缓存、浏览器缓存来应对，我将在系统优化章节详细介绍 CDN 缓存。

写热点的思路就比较简单了，我们共介绍了 3 种方法。一是本地缓存，延迟提交；二是将写热点数据进行分片，我们在处理大 key 时也经常用分片的思路；三是单 SKU 限流，保护单分片的 Redis 操作。

最后一部分我们介绍了系统容灾，容灾是解决系统级故障的手段，这是个比较大的话题，一般在互联网大厂中会有运维和架构组织统一设计方案，而不同公司选择的方案和路径不同，要结合你所在公司的具体情况而定。

需要特别注意的是，机房物理距离的问题，对你的方案设计至关重要，本质上难点就是数据的复制和一致性问题。这节课我们重点学习了“同城双活”的设计思路，通过秒杀系统的同城双活设计，你可以看到，不管是 Nginx 集群、Tomcat 集群、Redis 集群，还是 MySQL 集群，我们都可以灵活进行机房间切换，在故障时快速恢复。

思考题

这节课我们探讨了热点数据以及热点数据的几种处理方法，这里请你思考下，我们怎么能主动发现热点数据并进行预警呢？该如何设计？

以上就是这节课的全部内容，欢迎你在评论区和我讨论问题，交流经验！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 加餐 | 高并发场景：如何提升对突发事件的应急处理能力？

下一篇 09 | 御敌国门外：黑产对抗——防刷和风控

精选留言 (1)

写留言

一个卖火柴的老男人

2021-10-15

如饥似渴，太给力了！膜拜

展开